Modul 165

NoSQL-Datenbanken einsetzen

Inhaltsverzeichnis

1 Vorwort ... 3
2 Informieren ... 3
2.1 Projektumfang ... 3
2.2 Anforderungsanalyse ... 3
2.2.1 Technische Anforderungen ... 3
2.2.2 Funktionale Anforderungen ... 3
3 Planen ... 4
3.1 PSP ... 4
3.2 Datenbankdesign ... 5
3.2.1 Collections ... 5
3.2.2 Relations ... 6
3.3 API-Endpoints ... 7
4 Entscheiden ... 8
4.1 Wo liegt der Fokus? ... 8
5 Realisieren ... 8
5.1 Plugins installieren und konfigurieren ... 8
5.1.1 MongoDB-Driver ... 8
5.1.2 AutoMapper ... 8
5.1.3 JWT ... 8
5.1.4 Serilog ... 8
5.2 API-Projekt realisieren ... 9
5.3 Datenmigration ... 9
5.4 Backup- und Restore-Skript ... 10
5.5 Initialisierungsskript ... 10
5.6 Postman-Testprojekt ... 10
6 Kontrollieren ... 10
6.1 Postman ... 10
6.2 Swagger ... 10
7 Auswerten ... 11
7.1 Lessons Learned ... 11
7.2 Fazit ... 12
8 Abbildungsverzeichnis ... 13

1 Vorwort

Das vorliegende Dokument beschreibt die Entwicklung eines Backend-Systems und die Migration von Daten aus einer relationalen in eine NoSQL-Datenbank. Ziel war es, eine robuste Backend-Struktur zu schaffen, die eine sichere Authentifizierung mittels JWT bietet, strukturiert durch dedizierte Controller und Services für API-Endpoints, unterstützt durch eine Vielzahl von Modellen und Data Transfer Objects (DTOs) zur Optimierung des Datenaustausches. Die Implementierung nutzt MongoDB als Datenbank und ist vollständig in der GitHub-Repository dokumentiert.

Geplant und dokumentiert wird nach IPERKA.

2 Informieren

2.1 Projektumfang

Die Firma Jetstream-Service, ein mittelständisches Unternehmen, das sich auf Skiservicearbeiten während der Wintersaison spezialisiert hat, hat in den letzten Jahren signifikante Investitionen getätigt, um ihre digitalen Prozesse zu optimieren. Diese Investitionen umfassten die Entwicklung und Implementierung eines webbasierten Systems für die Auftragsanmeldung und -verwaltung, das auf einer robusten datenbankgestützten Plattform aufbaut. Aufgrund einer stetig wachsenden Auftragslage und der positiven Geschäftsentwicklung hat die Unternehmensleitung beschlossen, das Geschäftsfeld zu erweitern und an mehreren neuen Standorten Filialen zu eröffnen.

Die bisher verwendete relationale Datenbankarchitektur stößt jedoch angesichts dieser Expansion an ihre Grenzen, insbesondere was die Verteilung und Skalierung der Daten anbelangt. Die neuen geschäftlichen Herausforderungen erfordern eine flexible und skalierbare Datenmanagementlösung. Aus diesem Grund und um zukünftig hohe Lizenzkosten zu vermeiden, hat sich das Management dazu entschieden, die bestehende Datenbankinfrastruktur zu überdenken. Es wurde beschlossen, eine Migration von der derzeitigen relationalen Datenbank hin zu einem fortschrittlicheren und kosteneffizienteren NoSQL-Datenbanksystem zu vollziehen. Dieser Schritt soll nicht nur die technologische Basis des Unternehmens stärken, sondern auch sicherstellen, dass es den steigenden Anforderungen an das Datenmanagement gerecht wird und gleichzeitig eine solide Grundlage für zukünftiges Wachstum und Diversifizierung bildet.

2.2 Anforderungsanalyse

2.2.1 Technische Anforderungen

- Datenbankdesign und Implementierung
- Datenmigration von SQL in NoSQL
- Migration des bestehenden Backends
- Testprojekt in Postman

2.2.2 Funktionale Anforderungen

- Implementierung eines Mitarbeiter-Logins
- Mutation (Änderung) von Aufträgen
- Abruf und Anzeige der Serviceaufträge
- Möglichkeiten zum Löschen und Erstellen von Aufträgen
- Logging von wichtigen Aktivitäten und Fehlern in eine lokale Datei mit Serilog
- Filterung von Aufträgen nach Priorität

3 Planen

3.1 PSP

Nr.	Beschreibung	SOLL-Zeit	IST-Zeit
1	**Informieren**	**1.5**	**1.5**
1.1	Ausgangslage analysieren	0.5	0.5
1.2	Anforderungsanalyse	0.5	0.5
1.3	Benötigte Plugins in Erfahrung bringen	0.5	0..5
2	**Planen**	**3.5**	**4.0**
2.1	Erstellen eines Zeitplans	0.5	0.5
2.2	Datenbankdesign entwerfen	1.0	1.5
2.3	API Endpoints planen	1	1
3	**Entscheiden**	**0.5**	**0.5**
3.1	Wo liegt der Fokus?	0.5	0.5
4	**Realisieren**	**32.0**	
4.1	Plugins installieren und konfigurieren	3.0	2.5
4.2	API/Backend Projekt realisieren	5	5
4.3	Datenmigration	5	7
4.4	Backup- und Restore-Skripte schreiben	4	
4.5	Initialisierungsskript schreiben (auto. DB-Creation etc)	8	
4.6	Postman-Testprojekt erstellen	2	3.5
5	**Kontrollieren**	**2.0**	**2.0**
5.1	Postman-Testprojekt ausführen	0.5	0.5
5.2	API-Endpoints mit Swagger testen	1.5	1.5
6	**Auswerten**	**5.25**	**5.75**
6.1	Dokumentation nach IPERKA fertigstellen	3.5	4.0
6.2	Präsentation (PDF) erstellen	1.0	1.0
6.3	GitHub Repo vervollständigen	0.5	0.5
6.4	Abgabe des GitHub-Repo-Links	0.25	0.25
Gesamt		**37.25**	

3.2 Datenbankdesign

Das neue Datenbankdesign ist aufgebaut auf dem alten Design der relationalen Datenbank. Ich habe mir Gedanken gemacht, wie man die Daten am besten in eine NoSQL-Datenbank implementieren, sodass es den Zweck der Skalierbarkeit und Performance erfüllt.

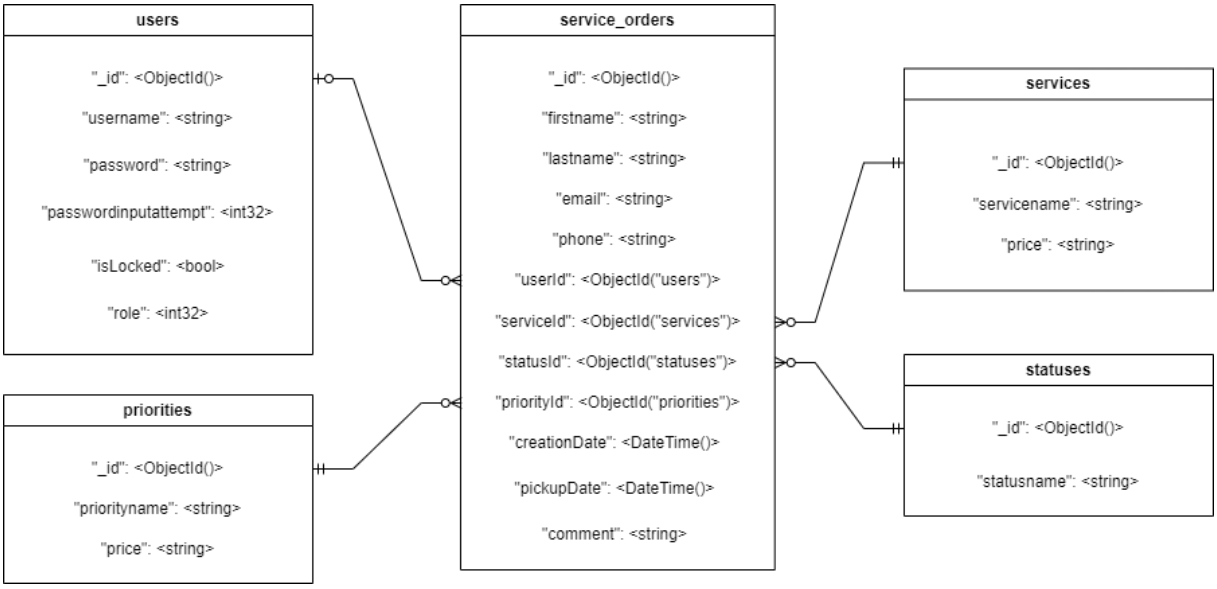

Abbildung 1 - Datenbankdesign

3.2.1 Collections

Users

Die Users-Collection speichert die Informationen der Mitarbeiter ab. Da haben wir ein «Username» und ein «Password», welche zum einloggen verwendet werden. Man bedenke, dass «Password» hier einfach nur so dargestellt wird – in der konkreten Datenbankstruktur besteht es aus einem Hash und einem Salt – wer schon mal Passwörter verschlüsselt hat weiss, was damit gemeint ist.

Ein Hash ist das Ergebnis einer Hash-Funktion, die aus einem Passwort einer beliebigen Länge einen Datensatz einer festen Länge erzeugt. Das macht es beinahe unmöglich, aus dem Hash dann das eigentliche Passwort retournieren zu können. Das Salt wird dann noch dazu gehängt, um die Sicherheit weiter zu erhöhen.

Dann gibt es noch einen «PasswordInputAttempt», welcher verwendet wird, um das Konto eines Mitarbeiters nach 3 Falscheingaben zu blockieren. Das Feld «IsLocked» definiert dann, ob der Account blockiert ist oder nicht.

Das «Role»-Field hingegen entspricht entweder einer 0 oder einer 1, je nachdem, ob der Account ein normaler Mitarbeiter oder ein Admin ist.

Priorities

Hier werden die davor definierten Prioritäten gespeichert – «Tief, Standard und Hoch».

Ausserdem haben sie unterschiedliche Preise, natürlich. Wer seine Skier nach 2 anstatt 7 Tagen bereits wieder haben möchte, muss draufzahlen.

Services
Hier das Selbe wie bei den Priorities. Hier werden die davor definierten Dienstleistungen, die das Unternehmen anbietet, hinterlegt.

Auch diese haben alle unterschiedliche Preise.

Statuses
Das Selbe in Grün – vordefinierte Zustände, entweder «Offen», «in Arbeit» oder «Abgeschlossen», je nachdem, wie der Stand eines Auftrags ist.

Dann gibt es noch den speziellen Status «Storniert». Hiermit kann ein Auftrag gekennzeichnet werden, der storniert wurde, ohne, dass man ihn direkt komplett aus der Datenbank entfernen muss. Gut für die jährlichen Bilanzen, die erstellt werden.

ServiceOrders
Das ist die Haupt-Collection. Hier laufen die Informationen aller Collections zusammen. Erstellt ein Kunde einen Auftrag, wird der als ServiceOrder hinterlegt.

Diverse Kundeninformationen wie «FirstName», «LastName», «Email» und «Phone» sind hier hinterlegt. Ausserdem die «UserId» des Mitarbeiters, der diesen Auftrag übernimmt – sobald jemand zugeteilt wurde natürlich. Darüber hinaus dann auch noch die «ServiceId», «StatusId» und «PriorityId», dazu jedoch mehr bei dem Punkt «Relations».

Ausserdem wird das «CreationDate» hinterlegt. Dieses spiegelt wieder, wann der Kunde den Auftrag erstellt hat. Das «PickupDate» definiert hingegen, wann der Kunde seine Ausrüstung wieder abgeholt hat.

Zu guter Letzt ein «Comment»-Field. Das ist nicht zwingend notwendig, aber dort kann ein Mitarbeiter einen Kommentar hinterlassen, den dann die anderen Mitarbeiter sehen können. Vielleicht braucht er das auch nur, um sich selber an etwas erinnern zu können. Diese Logik wurde jedoch nicht in den API-Endpoints umgesetzt.

3.2.2 Relations
Alle Collections haben eine Beziehung zu der «ServiceOrders»-Collection.
So wird dafür gesorgt, dass in der Datenbank kein Chaos herrscht, weil alles in einer einzelnen Collection gespeichert wird.

Hierbei handelt es sich jedoch um Informationen, die sonst immer und immer wieder neu gespeichert werden müssen, was die Sache sehr redundant machen würde. Daher ist es auch nur logisch, dass man diese über Relations zueinander verbindet.

3.3 API-Endpoints

Folgende API-Endpoints haben sich ergeben.
Diese umfassen die vom Kunden gewünschte Funktionalitäten. Diese Endpoints decken die Grundfunktionalitäten des Backend-Systems ab, einschließlich Authentifizierung, Datenmanipulation und Abrufen von Informationen.

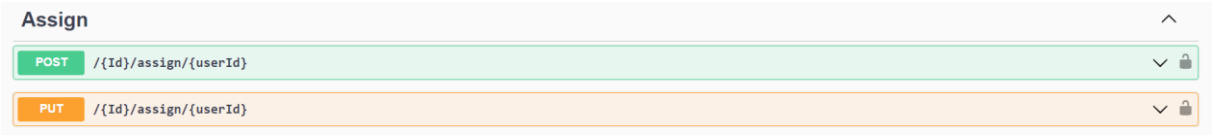

Abbildung 2 - Assign-API-Endpoints

Die Assign-Endpoints sind dafür da, um einen Mitarbeiter einem ServiceOrder zuzuweisen, welcher diesen dann bearbeitet. Jeder kann sich einem Auftrag zuweisen, doch nur ein Admin kann den zugewiesenen Mitarbeiter wieder ändern, wenn bereits einer zugewiesen ist.

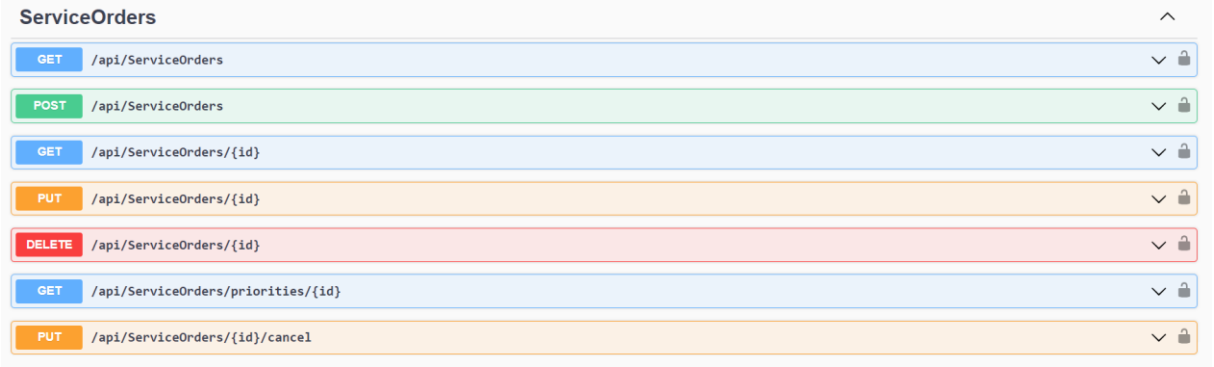

Abbildung 3 - ServiceOrders-API-Endpoints

Die ServiceOrders-Endpoints drehen sich komplett um die ServiceOrders, logischerweise. Egal ob man alle ServiceOrders getten will, einen neuen ServiceOrder erstellen will, einen bestimmten ServiceOrder anhand seiner ID getten, ändern, den Status auf «Storniert» oder ihn sogar löschen möchte oder alle ServiceOrders einer bestimmten Priority ausgeben möchte – damit sind keine Grenzen gesetzt.

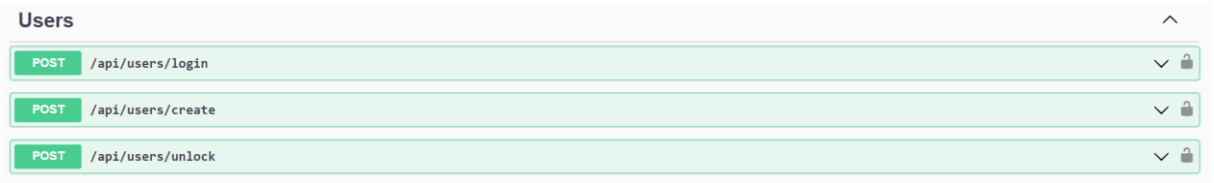

Abbildung 4 - Users-API-Endpoints

Die User-Endpoints sind für die Verwaltung der Mitarbeiter verantwortlich. Darüber kann man sich einloggen, wodurch man einen JWT erhält, den man wiederum bei einigen anderen Endpoints vorweissen muss, um die gewünschte Aufgabe durchführen zu können. Ausserdem kann man einen neuen Mitarbeiter-Account anlegen und gesperrte Accounts (weil zu oft das Passwort falsch eingegeben wurde) wieder entsperren – jedoch können diese beiden Operationen nur von einem Admin-Account erledigt werden.

4 Entscheiden

4.1 Wo liegt der Fokus?

Aufgrund der strengen Deadline und den hohen Anforderungen des Projekts macht es in meiner Situation Sinn, dass ich mir schon früh im Projekt Gedanken darüber mache, wo ich den Fokus setzen will, damit ich schlussendlich mindestens ein akzeptables Projekt abliefern kann.

Da ich wenig Erfahrung mit Skripte haben (weder in PowerShell, noch in Kombination mit MongoDB/mongosh), liegt mein Fokus nicht darauf. Zuerst muss alles andere stehen, bevor ich mich um diese kümmere.

Der Fokus liegt ganz klar in der Umstrukturierung der Backends, der Konfiguration von MongoDB und AutoMapper sowie in der Entwicklung einer vernünftigen und logischen Datenbank-Architektur.

5 Realisieren

5.1 Plugins installieren und konfigurieren

5.1.1 MongoDB-Driver

Da wir das Backend auf MongoDB umstellen müssen, brauchen wir den offiziellen MongoDB-Driver. Ein Nuget-Paket, welches mir die Interaktion mit der MongoDB ermöglicht. Vom Aufbau der Connection zur Datenbank, über die Definierung der spezifischen Collection, die ich für eine gewisse Aufgabe brauche, bis hin zu den einzelnen eigentlichen Interaktionen wie Daten getten, putten, deleten und posten.

5.1.2 AutoMapper

Dies ist beinahe schon das Kernstück der Umstellung zu MongoDB. Damit ich nicht alles umstellen muss, sondern meine Daten bequem zwischen Models und DTOs konvertieren kann, braucht es das Nuget-Paket AutoMapper.

Mit diesem kann man in einem MappingProfile gewisse Mappings erstellen wie das Mappen der Daten von einem Model in ein DTO und umgekehrt. Es macht die spätere Entwicklung einfacher, jedoch muss ich zugeben, dass ich so meine Schwierigkeiten damit hatte.

5.1.3 JWT

Wie bereits bei der SQL-Variante dieses API-Projekts, braucht es auch jetzt wieder eine Authentifizierung durch JWTs. Dementsprechend ist auch dieses Paket wieder vorhanden und wie benötigt konfiguriert.

5.1.4 Serilog

Auch ein Logging wird wieder vorausgesetzt. Das ist, wie bei der letzten Version, wieder mit Serilog realisiert. Zuverlässig, einfach in der Konfiguration und in der Bedienung.

5.2 API-Projekt realisieren

Die Backend-Entwicklung für das Skiservice-Projekt war eine anspruchsvolle Aufgabe, die sorgfältige Planung und Durchführung erforderte. Der Prozess begann mit der Einrichtung des ASP.NET WebAPI-Projekts, das als Grundlage für die gesamte Anwendung diente. Besonders wichtig war die Einbindung einer NoSQL-Datenbank. Diese Entscheidung ermöglichte eine effiziente und flexible Handhabung der Daten, wobei die Datenbankstruktur sorgfältig auf die Bedürfnisse des Projekts abgestimmt wurde, um eine von Zeit zu Zeit passierende Skalierung garantieren zu können, ohne grossen administrativen und finanziellen Aufwand zu verursachen.

Ein kritischer Aspekt der Backend-Entwicklung war die Implementierung eines sicheren Authentifizierungssystems mittels JWT (JSON Web Tokens). Dies war entscheidend, um den Mitarbeitern einen sicheren und autorisierten Zugriff auf die Serviceaufträge zu ermöglichen. Die Konfiguration und Verwaltung der Token war eine interessante und dennoch selbsterklärende Aufgabe.

Die Erstellung der verschiedenen API-Endpunkte war ein weiterer wichtiger Schritt. Jeder Endpunkt wurde sorgfältig entwickelt, um spezifische Funktionen wie das Anlegen, Abrufen, Ändern und Löschen von Serviceaufträgen sowie das Filtern der Aufträge nach Priorität zu ermöglichen. Hierbei wurde besonderes Augenmerk auf die Business-Logik gelegt, um sicherzustellen, dass die Endpunkte die Anforderungen des Projekts effizient erfüllen.

Um die Stabilität und Zuverlässigkeit der API zu gewährleisten, wurden robuste Fehlerbehandlungssysteme und Logging-Mechanismen implementiert. Das Logging erfolgte in lokale Dateien, was eine wichtige Rolle bei der Fehlerbehebung und der Überwachung der Systemaktivitäten spielte. Die Fähigkeit, Ereignisse zu protokollieren und aufzuzeichnen, war entscheidend für die Wartung und den Support der Anwendung.

5.3 Datenmigration

Um eine Datenmigration und im allgemeinen die Umstellung von SQL auf NoSQL zu ermöglichen, ohne das ganze Projekt umschreiben oder neu schreiben zu müssen, muss sorgfältig und bedacht gehandelt werden.

Durch den AutoMapper konnte die Kommunikation und der Transfer der Daten zwischen den Modellen und den DTOs (Data Transfer Objects) gewährleistet werden. Die Modelle und DTOs mussten nur geringfügig an die etwas veränderte Datenbankstruktur der NoSQL-Datenbank im Vergleich zu der SQL-Datenbank angepasst werden. Die Calls in den APIs funktionieren nun etwas anders, da es nicht mehr über das EF-Core läuft, dennoch sind sich die Abläufe und die Strukturen ähnlich.

5.4 Backup- und Restore-Skript
Ich habe es versucht, leider hat mir die Zeit nicht gereicht, um dies vollenden zu können…

5.5 Initialisierungsskript
Ich habe es versucht, leider hat mir die Zeit nicht gereicht, um dies vollenden zu können…

5.6 Postman-Testprojekt
Die Aufgabe mit dem Postman-Testprojekt war sehr spannend. Beim letzten Modul wurde eine einfache Collection erstellt, mit der man Schritt für Schritt jeden Endpoint mit benutzerdefinierten Daten füttern und so testen konnte, ob die API-Endpoints so funktionieren, wie sie es sollten.

Nun wurden noch zusätzlich über 50 Tests erstellt. Diese können durch einen einzigen Klick ausgeführt werden. Das erleichtert das Testing, verlängert jedoch auch die Erstellungszeit signifikant.

6 Kontrollieren
6.1 Postman
Da das Postman-Testprojekt nun nicht nur eine Collection der API-Endpoints ist, sondern auch über 50 Tests beinhaltet, konnte das Testing schnell und sauber erfüllt werden.

Alle Tests werden durch einen Klick auf einen Button nach und nach ausgeführt und zeigen alle Tests als «passed» an. Ein Export dieses durchgeführten Tests wurde im Verzeichnis «./Files/Postman/ExportedResults» hinterlegt.

6.2 Swagger
Das ist die im Projekt integrierte Variante des Testings – Swagger. Wird das Projekt gestartet, so öffnet sich direkt die Swagger-Homepage. Dort werden alle API-Endpoints aufgelistet und man kann sie nach und nach mit verschiedenen Inputs testen. Für einige wird ein JWT benötigt, welchen man sich auch durch die Benutzung des Login-API-Endpoints besorgen kann.

Alle Tests verliefen ohne Probleme und lieferten das gewünschte Ergebnis.

7 Auswerten

7.1 Lessons Learned

Meine Vorliebe für NoSQL-Datenbanken, insbesondere für MongoDB-Varianten wie Firebase, AppWrite und ähnliche, ist aus vielerlei Hinsicht stark ausgeprägt. Diese modernen Datenbanklösungen bieten eine Flexibilität und Skalierbarkeit, die in der heutigen schnelllebigen digitalen Welt unerlässlich ist. Sie ermöglichen es, mit nicht-relationalen Datenstrukturen zu arbeiten, was oft zu einer Vereinfachung der Entwicklung und Wartung von Anwendungen führt.

Allerdings musste ich feststellen, dass die Migration eines bestehenden Projekts, das auf einem relationalen SQL-Datenbanksystem basiert, zu einem NoSQL-basierten System weit komplexer ist, als ich ursprünglich angenommen hatte. Die Annahme, dass die Migration relativ unkompliziert sein würde, beruhte auf meiner umfangreichen Erfahrung mit NoSQL-Datenbanken in meinen eigenen Projekten. Die Realität hat mich jedoch eines Besseren belehrt.

Die Herausforderungen, denen ich mich stellen musste, waren vielschichtiger als erwartet. Dinge, bei denen ich dachte, dass diese nicht schwierig seien, haben mich dann doch teilweise beinahe überfordert. Diese Tatsache hat mich oft an den Rand der Frustration gebracht und dazu geführt.

Die Erfahrung, die ich während dieses Migrationsprozesses sammelte, war zwar lehrreich, aber auch psychisch belastend. Es hat mir die Augen dafür geöffnet, wie anspruchsvoll und komplex Datenmigrationen tatsächlich sein können. Es war eine Lektion in Demut und hat mich dazu veranlasst, meine eigene Expertise kritisch zu hinterfragen. Der Gedanke, dass ich möglicherweise in meiner zukünftigen Karriere erneut mit einer ähnlichen Aufgabe konfrontiert werden könnte, lässt mich innerlich zurückschrecken. Ich erinnere mich dann an das Abschlussprojekt und stelle mir vor, wie ich, um die Belastung zu bewältigen, möglicherweise professionelle Hilfe in Anspruch nehmen müsste, vielleicht sogar einen Termin bei einem Psychiater vereinbaren müsste, um die Erfahrungen aufzuarbeiten (das ist natürlich überspitzt formuliert).

Trotz der Annahme, in einem bestimmten Bereich bereits tiefgreifende Kenntnisse zu besitzen, wurde mir klar, dass es immer noch viel zu lernen gibt. Die Informatik ist in der Tat ein Bereich des lebenslangen Lernens, und diese Erfahrung hat diese Erkenntnis nur noch verstärkt. Es hat mich demütig gemacht und mir gezeigt, dass unabhängig davon, wie viel Erfahrung man glaubt zu haben, stets neue Horizonte zu entdecken sind, die es zu erforschen gilt. Diese Erkenntnis ist wertvoll und treibt mich an, stets offen für neues Wissen und neue Herausforderungen in diesem sich ständig entwickelnden Feld zu sein.

7.2 Fazit

Nach intensiver Arbeit und beträchtlicher Anstrengung bin ich mit dem Ergebnis dieses Projekts im Grossen und Ganzen sehr zufrieden. Die Transformation der Datenbankarchitektur von einem SQL-basierten zu einem NoSQL-basierten System wurde erfolgreich umgesetzt, und die Mehrheit der Projektanforderungen wurde adäquat erfüllt. Diese Umstellung, obwohl technisch gelungen, war jedoch mit einer Reihe von Herausforderungen verbunden, die mich an die Grenzen meiner Geduld und meines technischen Know-hows gebracht haben.

Trotz des erfolgreichen Abschlusses des Projekts muss ich gestehen, dass meine Affinität zu der verwendeten Programmiersprache nicht besonders stark ist. Es ist ein Gefühl, das schwer zu artikulieren ist, denn wenn ich an andere Sprachen wie Rust, Java (obwohl sehr ähnlich) und Go beispielsweise denke, erlebe ich nicht dieselben emotionalen Turbulenzen.

Das Projekt hat mich auf eine Weise gefordert, die ich bisher selten erlebt habe. Es war eine mentale und technische Herausforderung, die gelegentlich an meiner Ausdauer gezehrt hat. Kopfschmerzen waren keine Seltenheit, und es gab Momente, in denen ich mich fragte, ob ich die richtige Vorgehensweise gewählt hatte. Doch trotz aller Widrigkeiten habe ich das Projekt zu einem erfolgreichen Abschluss gebracht.

Rückblickend gibt es natürlich Aspekte, die ich gerne noch verfeinert hätte. Wenn ich die Zeit zurückdrehen könnte, würde ich mir ein paar zusätzliche Tage gönnen, um die letzten Feinheiten zu perfektionieren und einige Aspekte des Projekts zu verbessern. Ich finde mich jedoch immer in dem Wunsch nach mehr Zeit wieder, einem Gefühl, das wahrscheinlich vielen Kreativen und Entwicklern bekannt ist. Es scheint eine universelle Wahrheit zu sein, dass unabhängig davon, wie viel Zeit einem für ein Projekt zur Verfügung steht, der Wunsch besteht, nur ein bisschen mehr davon zu haben, um das Werk zu verfeinern.

Jetzt, da das Projekt abgeschlossen ist, fühle ich eine Mischung aus Erleichterung und Nachdenklichkeit. Ich bin erleichtert, dass ich diese Aufgabe hinter mir habe, und gleichzeitig reflektiere ich über das, was hätte sein können, wenn ich nur ein wenig mehr Zeit gehabt hätte. Diese Reflexion ist nicht nur ein Teil des Abschlussprozesses, sondern auch ein essentieller Teil des Lernprozesses, der mich als Fachperson und als Individuum wachsen lässt. Ich nehme diese Erfahrungen mit, sowohl die guten als auch die herausfordernden, und weiß, dass sie für zukünftige Projekte von unschätzbarem Wert sein werden.

8 Abbildungsverzeichnis

Abbildung 1 - Datenbankdesign 5
Abbildung 2 - Assign-API-Endpoints 7
Abbildung 3 - ServiceOrders-API-Endpoints 7
Abbildung 4 - Users-API-Endpoints 7